

FIGURE 1

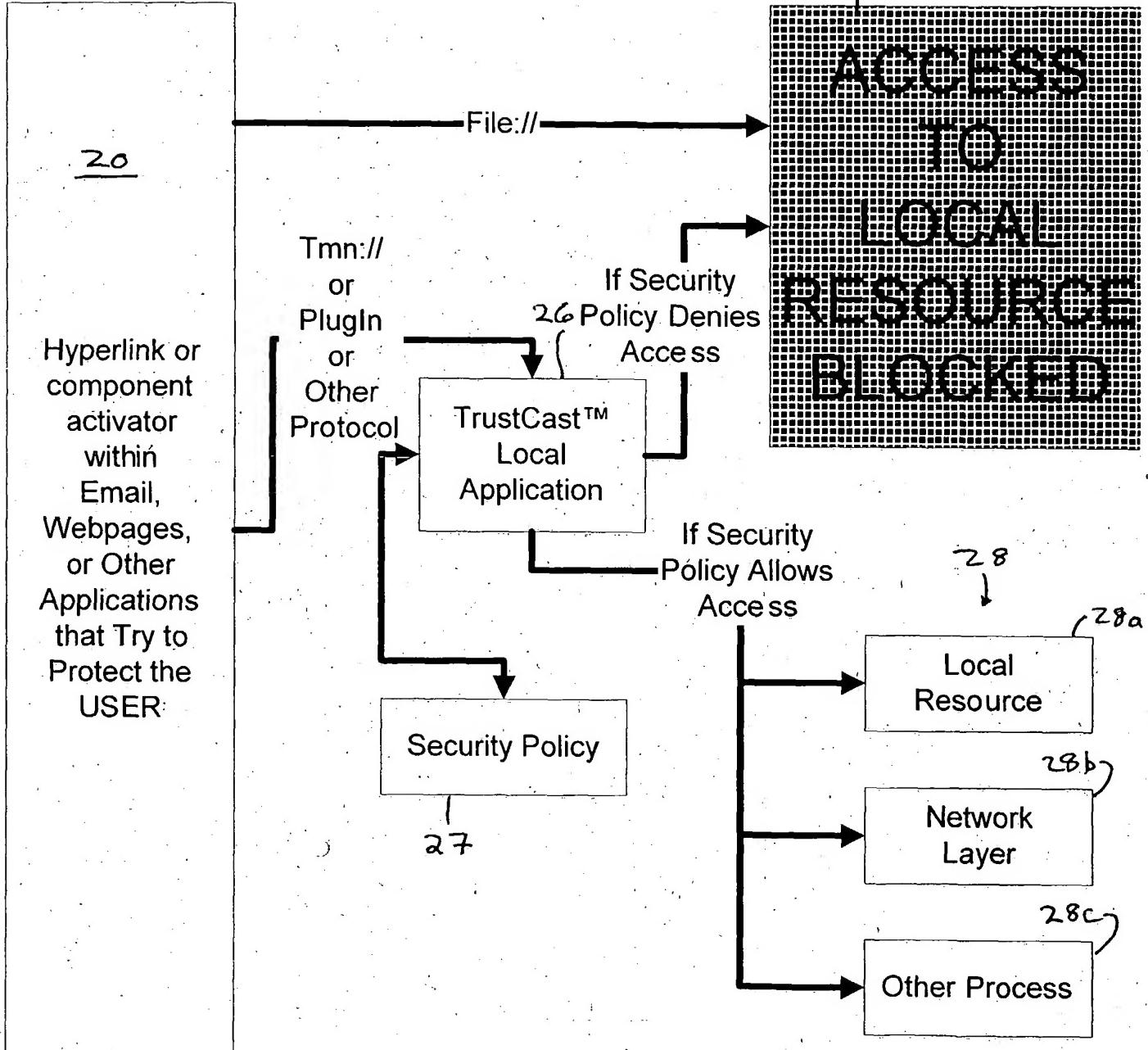


FIGURE 1A

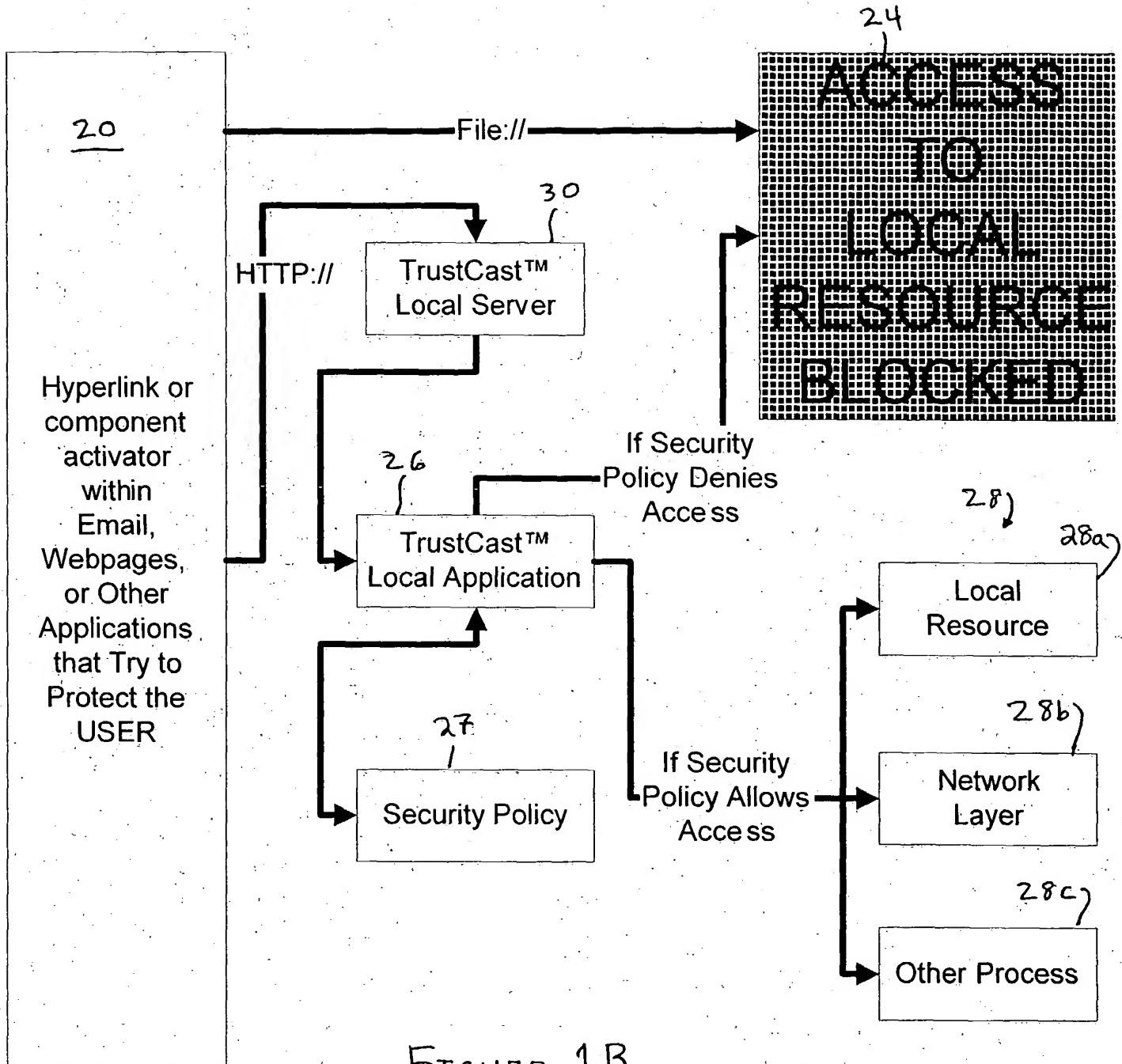
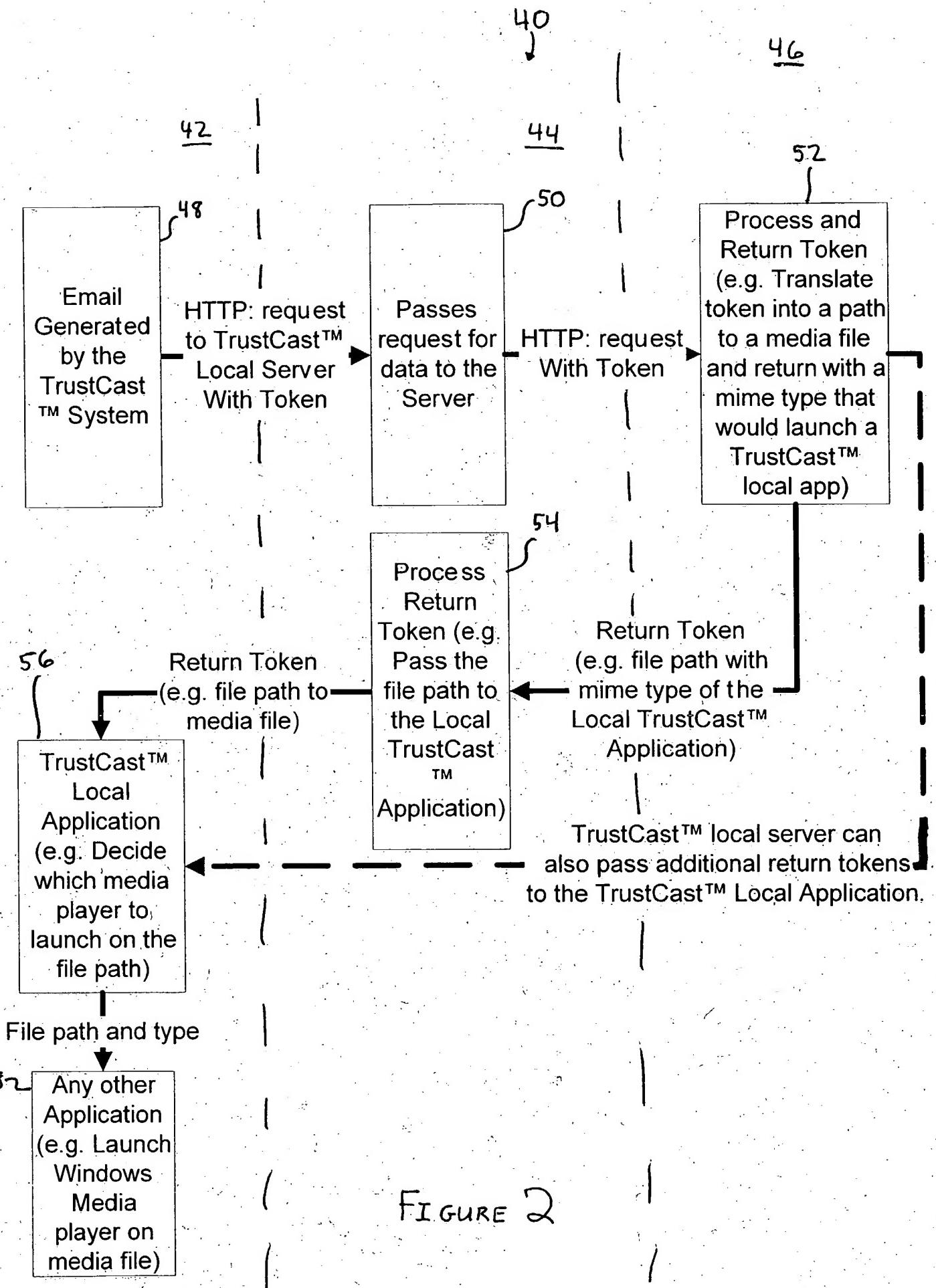


FIGURE 1B



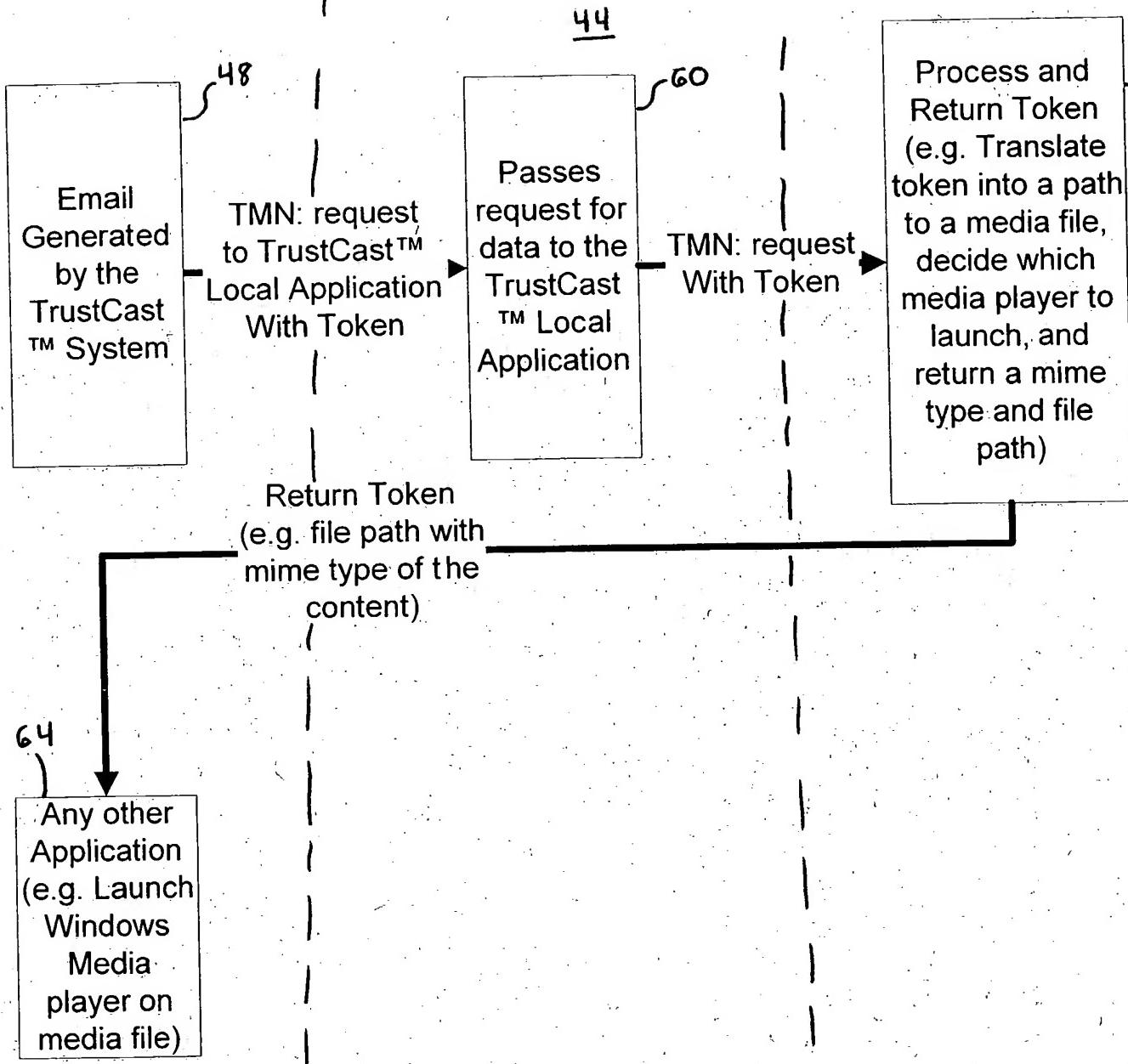


FIGURE 2A

```

----- the osx preferred embodiment
def ConstructBrowser(name, agent, style):
    b = {}
    b['name'] = name      # Pretty name
    b['agent'] = agent    # Identifying UserAgent substring
    b['style'] = style    # Supported link styles

    return b

def ConstructMailer(name, format, style):
    m = {}
    m['name'] = name      # Pretty name
    m['format'] = format  # Supported delivery formats
    m['style'] = style    # Supported link styles

    return m

def ConstructOSXData():
    osx = {}
    webmails = {}
    webmails['AOL'] = ConstructWebmail('AOL', 'aol.com', (multipart,), (protocol, loopback, localhost,))
    webmails['CS'] = ConstructWebmail('CompuServe', 'cs.com', (multipart,), (attachment,))
    webmails['ATT'] = ConstructWebmail('ATT Worldnet', 'worldnet.com', (text,), (protocol, loopback, localhost, attachment,))
    webmails['ATT'] = ConstructWebmail('Earthlink', 'earthlink.com', (text,), (attachment,))
    webmails['Hot'] = ConstructWebmail('Hotmail', 'hotmail.com', (multipart,), (loopback, localhost,))
    webmails['Yah'] = ConstructWebmail('Yahoo!', 'yahoo.com', (multipart,), (loopback, localhost,))
    osx['webmails'] = webmails

    mailers = {}
    mailers['AOp3'] = ConstructMailer('AOL 8', (multipart,), ())
    mailers['emal'] = ConstructMailer('Apple Mail', (multipart,), (protocol, loopback, localhost,))
    mailers['OPIM'] = ConstructMailer('Entourage X', (multipart,), (loopback, localhost, file,))


```

FIGURE 2B1

```

        mailers['CSOm'] = ConstructMailer('Eudora 5+',  

        (multipart,), (protocol, loopback, localhost,))  

        mailers['Mgll'] = ConstructMailer('Magellan',  

        (localhost, loopback, file,)) (text,)  

        mailers['BLTO'] = ConstructMailer('Mailsmith',  

        (localhost, loopback, file,)) (text,)  

        mailers['Mlby'] = ConstructMailer('Mulberry',  

        (localhost, loopback, file, attachment,)) (text,)  

        mailers['Cm1t'] = ConstructMailer('PowerMail',  

        (multipart,), (protocol, loopback, localhost,))  

        mailers['CeLP'] = ConstructMailer('QuickMail Pro',  

        (text,), (protocol, loopback, localhost,))  

        osx['mailers'] = mailers

        browsers = {}  

        browsers['AOp3'] = ConstructBrowser('AOL 8',  

        'AOL', (protocol,))  

        browsers['CHIM'] = ConstructBrowser('Camino',  

        'Camino', (protocol, file,))  

        browsers['iCAB'] = ConstructBrowser('iCab 2.9+',  

        'iCab', (protocol, loopback, localhost, attachment,))  

        browsers['MSIE'] = ConstructBrowser('Internet Explorer 5.2+',  

        'MSIE  
5.2', (loopback, localhost, file, attachment,))  

        browsers['MOZZ'] = ConstructBrowser('Mozilla 1.3+',  

        'fnord', (protocol, loopback, localhost, file, attachment,))  

        browsers['MOSS'] = ConstructBrowser('Netscape Navigator 7+',  

        'Netscape', (protocol,))  

        browsers['OWEB'] = ConstructBrowser('OmniWeb 4.1+',  

        'OmniWeb', (protocol, file,))  

        browsers['OPRA'] = ConstructBrowser('Opera 6',  

        'Opera', (protocol, loopback, localhost, attachment,))  

        browsers['sfri'] = ConstructBrowser('Safari 1.0 v74 (aka Beta 2)',  

        'Safari', (protocol, file,))  

        osx['browsers'] = browsers

        return osx
def ConstructMacData():
    mac = {}

    webmails = {}

```

FIGURE 2B2

```
    webmails['AOL'] = ConstructWebmail('AOL',      'aol.com',
(multipart,),      (protocol, loopback, localhost,))
    webmails['CS'] = ConstructWebmail('CompuServe', 'cs.com',
(multipart,),      (attachment,))
    webmails['ATT'] = ConstructWebmail('ATT Worldnet', 'worldnet.com',
(text,),          (protocol, loopback, localhost, attachment,))
    webmails['ATT'] = ConstructWebmail('Earthlink',   'earthlink.com',
(text,),          (attachment,))
    webmails['Hot'] = ConstructWebmail('Hotmail',    'hotmail.com',
(multipart,),     (loopback, localhost,))
    webmails['Yah'] = ConstructWebmail('Yahoo!',     'yahoo.com',
(multipart,),     (loopback, localhost,))
    mac['webmails'] = webmails
```

```
    mailers = {}
    mailers['AOp3'] = ConstructMailer('AOL 5',
(multipart,),      ())
    mailers['MSNM'] = ConstructMailer('Outlook Express',
(multipart,),      (protocol, loopback, localhost, file,))
    mailers['OPIM'] = ConstructMailer('Entourage 2001',
(multipart,),      (protocol, loopback, localhost, file, attachment,))
    mailers['CSOm'] = ConstructMailer('Eudora',
(multipart,),      (protocol, loopback, localhost, file,))
    mac['mailers'] = mailers
```

```
    browsers = {}
    browsers['AOp3'] = ConstructBrowser('AOL 5',
'AOL',            (protocol, file, attachment,))
    browsers['iCAB'] = ConstructBrowser('iCab 2.9+',
'iCab',           (protocol, loopback, localhost, file,))
    browsers['MSIE'] = ConstructBrowser('Internet Explorer',
'MSIE',           (protocol, loopback, localhost, file, attachment,))
    browsers['MOZZ'] = ConstructBrowser('Mozilla 1.2.1',
'fnord',          (protocol, loopback, localhost, file, attachment,))
    browsers['MOSS'] = ConstructBrowser('Netscape Navigator 6',
'Netscape',        (protocol, loopback, localhost, attachment,))
    browsers['OPRA'] = ConstructBrowser('Opera 5',
'Opera',          (protocol, file,))
    mac['browsers'] = browsers
```

FIGURE 2B3

```

return mac

def ConstructWebmail(name, domain, format, style):
    w = {}
    w['name'] = name      # Pretty name
    w['domain'] = domain  # Root domain
    w['format'] = format  # Supported delivery formats
    w['style'] = style    # Supported delivery styles

    return w

def DetermineMailerSettings(_mailto,_http,pd):
    error = {}

    mailers = pd['mailers']
    browsers = pd['browsers']

    # Is the mailer supported?
    if _mailto not in mailers.keys():
        error['field'] = 'mailer'
        error['value'] = _mailto
        error['message'] = 'Sorry, that Email Client is not supported.'
        error[error['field']] = mailers.keys()
        return error

    # Is there a standalone link style?
    for l in (protocol,):
        if l in mailers[_mailto]['style']:
            error['format'] = mailers[_mailto]['format'][0]
            error['style'] = l
            return error

    # Does it work with the browser?
    if _http not in browsers.keys():
        error['field'] = 'browser'
        error['value'] = _http
        error['message'] = 'Sorry, that Web Browser is not supported.'
        error[error['field']] = browsers.keys()
        return error

```

FIGURE 2B4

```

for l in mailers[_mailto]['style']:
    if l in browsers[_http]['style']:
        error['format'] = mailers[_mailto]['format'][0]
        error['style'] = l
        return error

# Nope, so suggest one that will.
error['field'] = 'browser'
error['value'] = _http
error['message'] = 'Sorry, that Web Browser does not work with your Email Client.'

error[error['field']] = []
for l in mailers[_mailto]['style']:
    for _http in browsers.keys():
        if l in browsers[_http]['style']:
            error[error['field']].append(_http)
return error

def DetermineWebmailSettings(inProvider, _http,pd):
    error = {}
    webmails = pd['webmails']
    mailers = pd['mailers']
    browsers = pd['browsers']
    # Is the webmail system supported?
    if inProvider not in webmails.keys():
        error['field'] = 'provider'
        error['value'] = inProvider
        error['message'] = 'Sorry, that Email Provider is not a supported Web Mail system.'
        error[error['field']] = webmails.keys()
        return error

    # Does it work with the browser?
    if _http not in browsers.keys():
        error['field'] = 'browser'
        error['value'] = _http
        error['message'] = 'Sorry, that Web Browser is not supported.'
        error[error['field']] = browsers.keys()
        return error

    for l in webmails[inProvider]['style']:

```

FIGURE 2B5

```

if l in browsers[_http]['style']:
    error['format'] = webmails[inProvider]['format'][0]
    error['style'] = 1
    return error

# Nope, so suggest one that will.
error['field'] = 'browser'
error['value'] = _http
error['message'] = 'Sorry, that Web Browser does not work with your Web
Mail system.'
error[error['field']] = []
for l in webmails[inProvider]['style']:
    for _http in browsers.keys():
        if l in browsers[_http]['style']:
            error[error['field']].append(_http)

return error
def DetermineDeliverySettings(inProvider,_mailto,_http,pd):
    if inProvider != 'ISP':
        return
DetermineWebmailSettings(inProvider=inProvider,_http=_http,pd=pd)

    return DetermineMailerSettings(_mailto=_mailto,_http=_http,pd=pd)
def ProviderFromEmail(inEmail,pd):
    webmails = pd['webmails']
    atIndex = string.find(inEmail,'@')
    domain = inEmail[atIndex + 1:]
    for w in webmails.keys():
        if webmails[w]['domain'] == domain:
            return w
    return 'ISP'
def PlatformDeliveryValue(field,pd,inEmail,_http=None,_mailto=None):
    provider = ProviderFromEmail(inEmail=inEmail,pd=pd)
    error =
DetermineDeliverySettings(inProvider=provider,_mailto=_mailto,_http=_http,pd=
pd)

    if error.has_key(field):
        return error[field]

```

FIGURE 2B6

```

return None
----- the generalized code path
## file, localhost, loopback, attachment
def GetDefaultDeliveryStyle(self, platform = None):
    mappings = { 'mac' : 'protocol',
                 'osx' : 'localhost'}
    if platform in mappings.keys():
        delivery_style = mappings[platform]
    else:
        delivery_style = self.roots['DefaultDeliveryStyle']
    return delivery_style
## plain, html, multi
def GetDefaultDeliveryFormat(self, platform = None):
    return self.roots['DefaultDeliveryFormat']

def DeliveryFormat(self, recipient, _os = None, _http = None, _mailto = None):
    if self.platform_helpers.has_key(_os):
        delivery_format =
PlatformDeliveryValue(field='format',pd=self.platform_helpers[_os],inEmail=recipient,_mailto=_mailto,_http=_http)
        if delivery_format:
            return delivery_format

    mappings = {'earthlink.com' : 'plain'}
    delivery_format = self.GetDefaultDeliveryFormat(_os)
    index = recipient.rfind('@')
    mail_host = recipient[index+1:]
    if mail_host.lower() in mappings.keys():
        delivery_format = mappings[mail_host]
    return delivery_format

def DeliveryStyle(self, recipient, _os = None, _http = None, _mailto = None):
    if self.platform_helpers.has_key(_os):
        delivery_style =
PlatformDeliveryValue(field='style',pd=self.platform_helpers[_os],inEmail=recipient,_mailto=_mailto,_http=_http)
        if delivery_style:
            return delivery_style

```

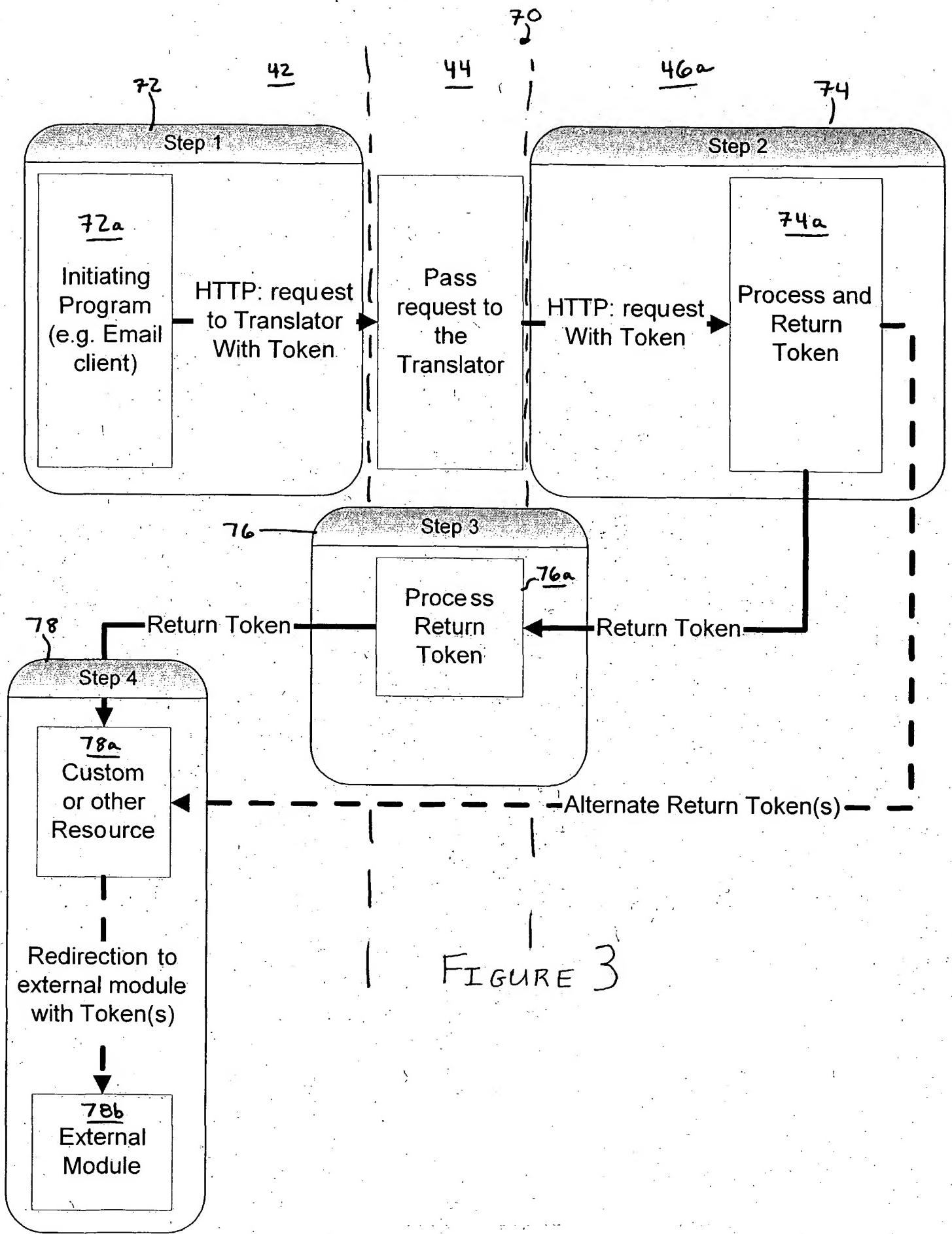
FIGURE 2B7

```
## lower case ONLY
mappings = { None : {'earthlink.com' : 'attachment', 'cs.com' : 'attachment',
'aol.com' : 'loopback'},
'mac' : {'cs.com' : 'attachment', 'aol.com' : 'file', 'yahoo.com' :
'localhost'},
'osx' : {'cs.com' : 'attachment', 'aol.com' : 'file', 'yahoo.com' :
'localhost'},
'win' : {'earthlink.com' : 'attachment', 'cs.com' : 'attachment', 'aol.com' :
'loopback'} }
delivery_style = self.GetDefaultDeliveryStyle(_os)
index = recipient.rfind('@')
if _os in mappings.keys():
    os_mappings = mappings[_os]
    mail_host = recipient[index+1:].lower()
    if mail_host in os_mappings.keys():
        delivery_style = os_mappings[mail_host]
return delivery_style
```

----- an xml fragment used in conjunction w/ code above

```
<key>DefaultDeliveryFormat</key>
<string>multi</string>
<key>DefaultDeliveryStyle</key>
<string>localhost</string>
```

FIGURE 2B8



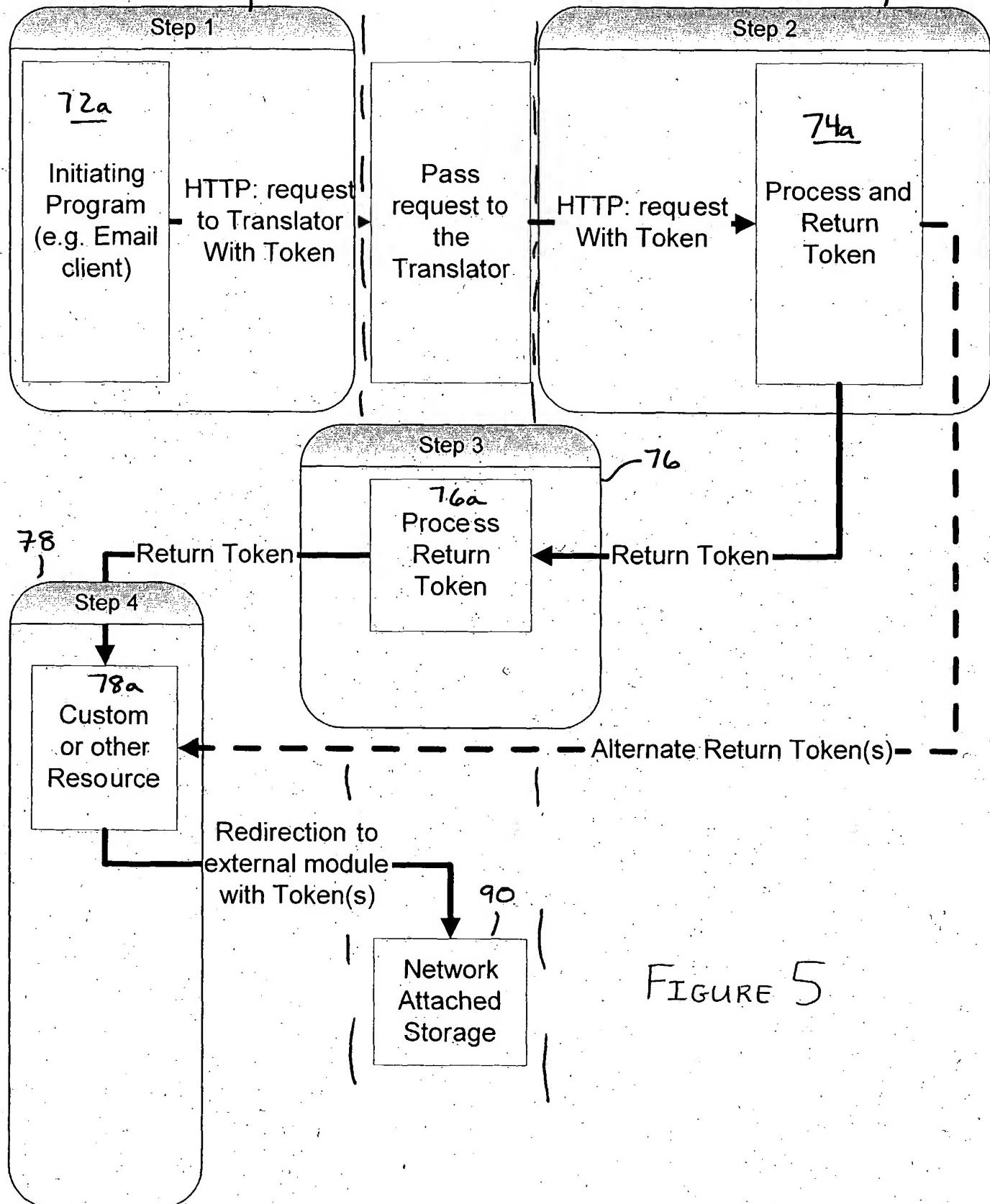


FIGURE 5

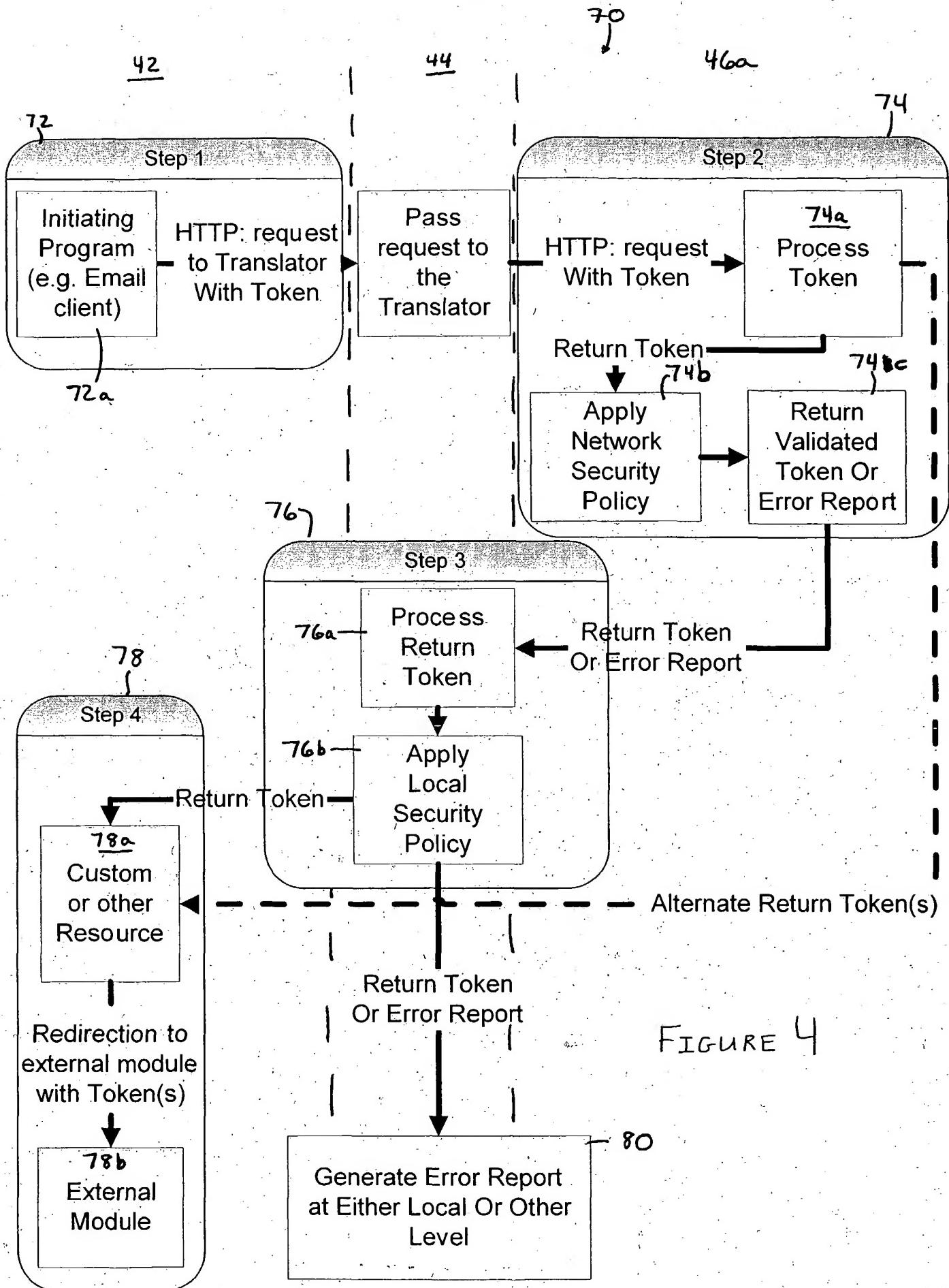
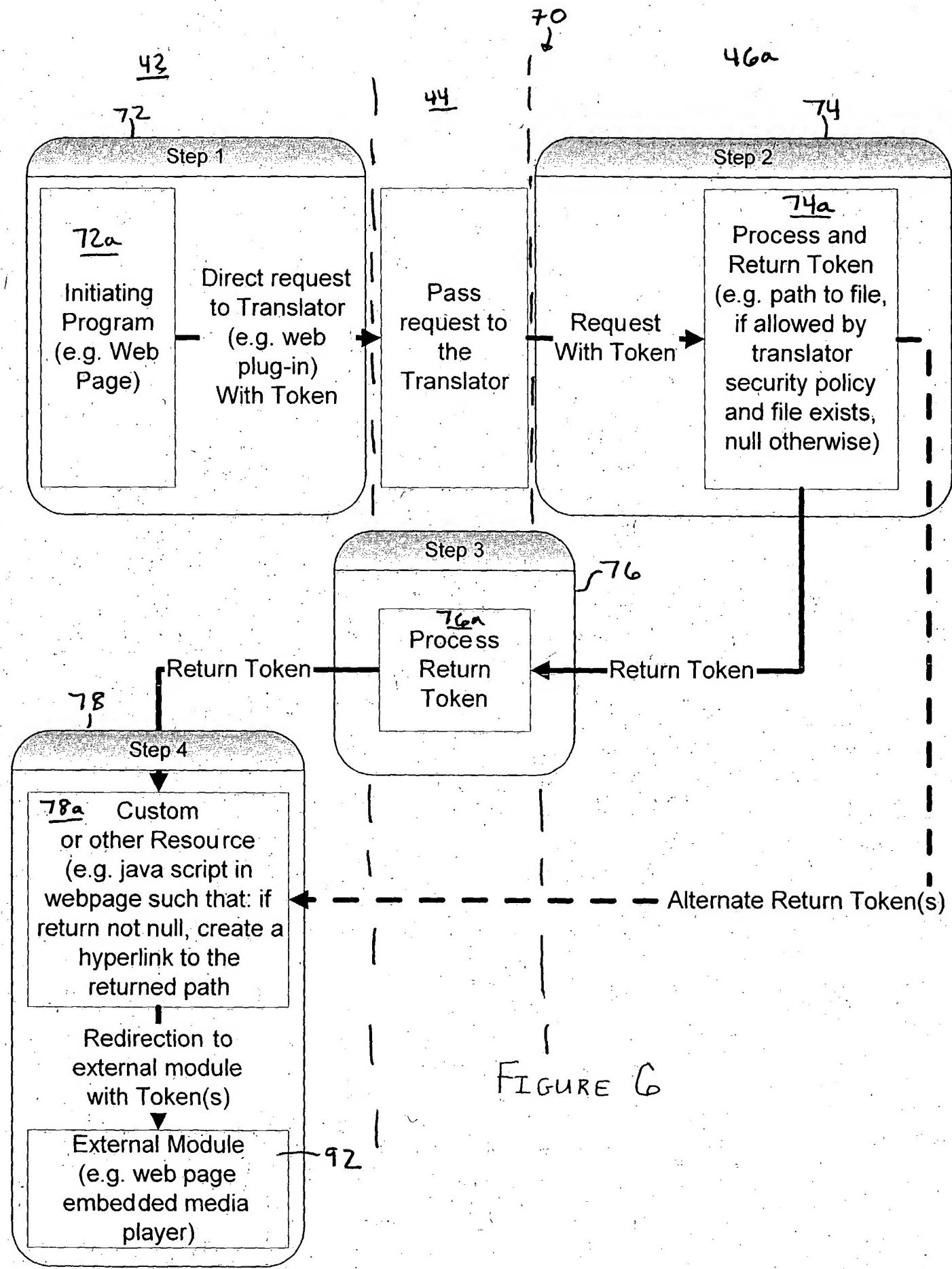


FIGURE 4

80



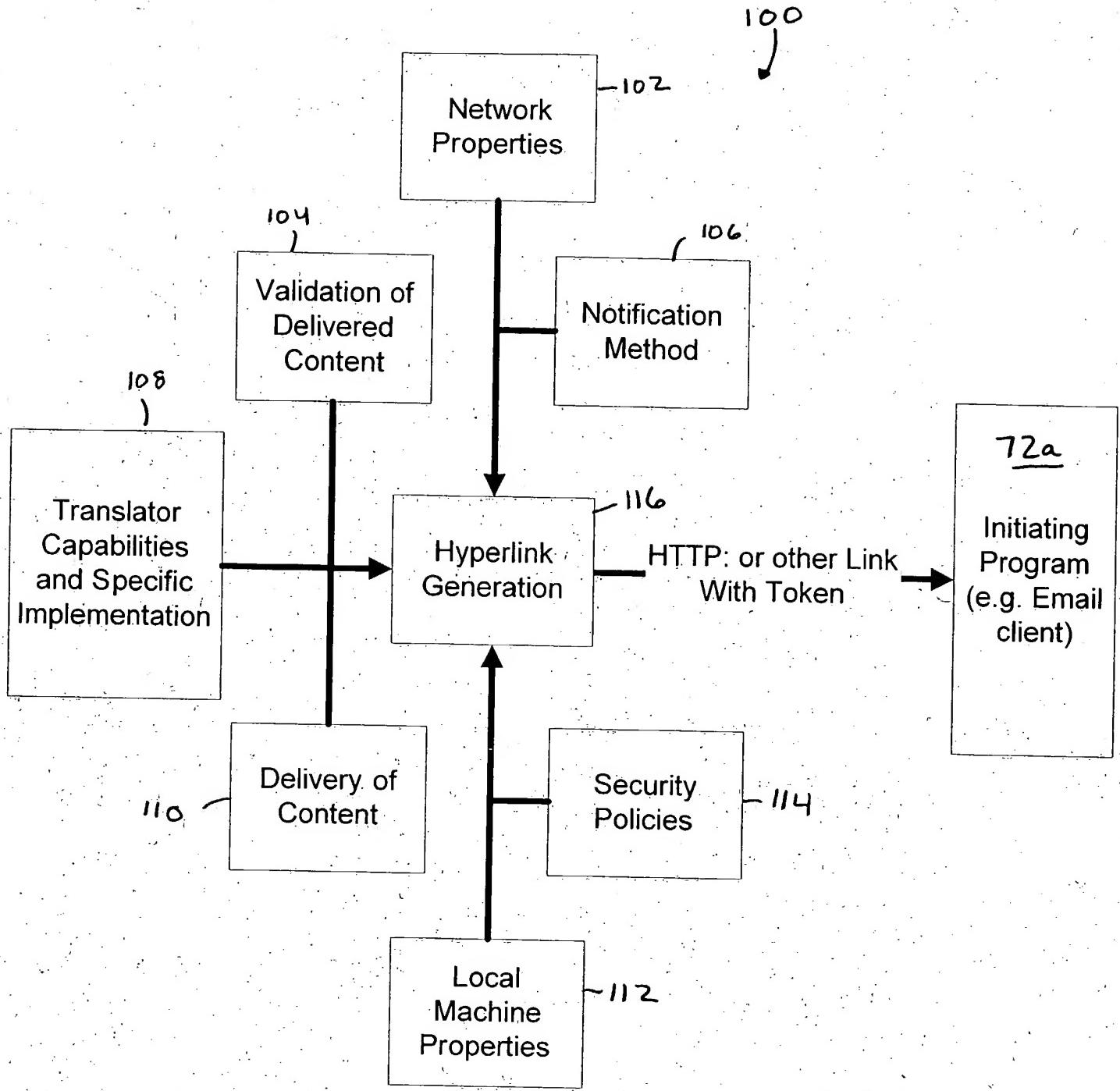
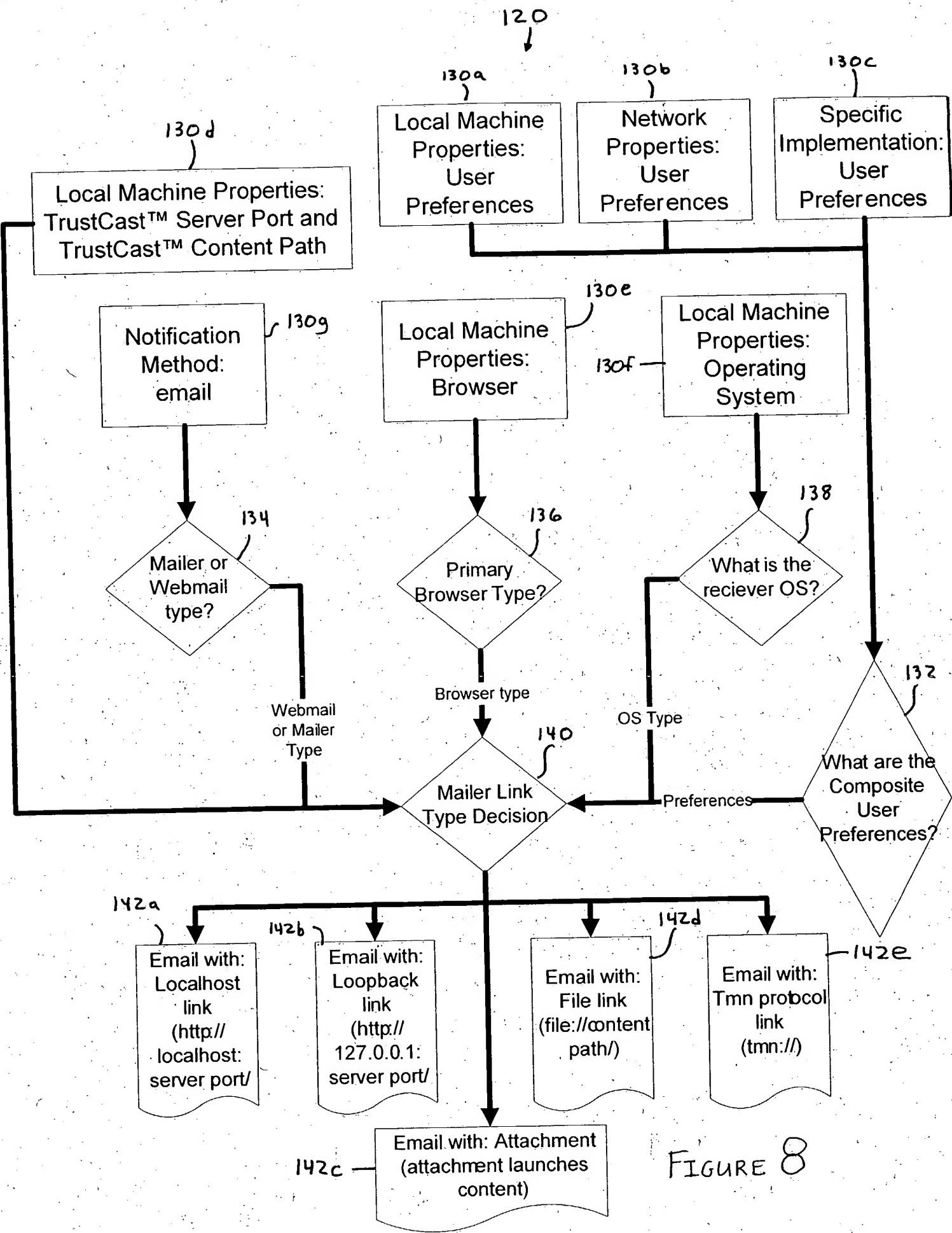


FIGURE 7



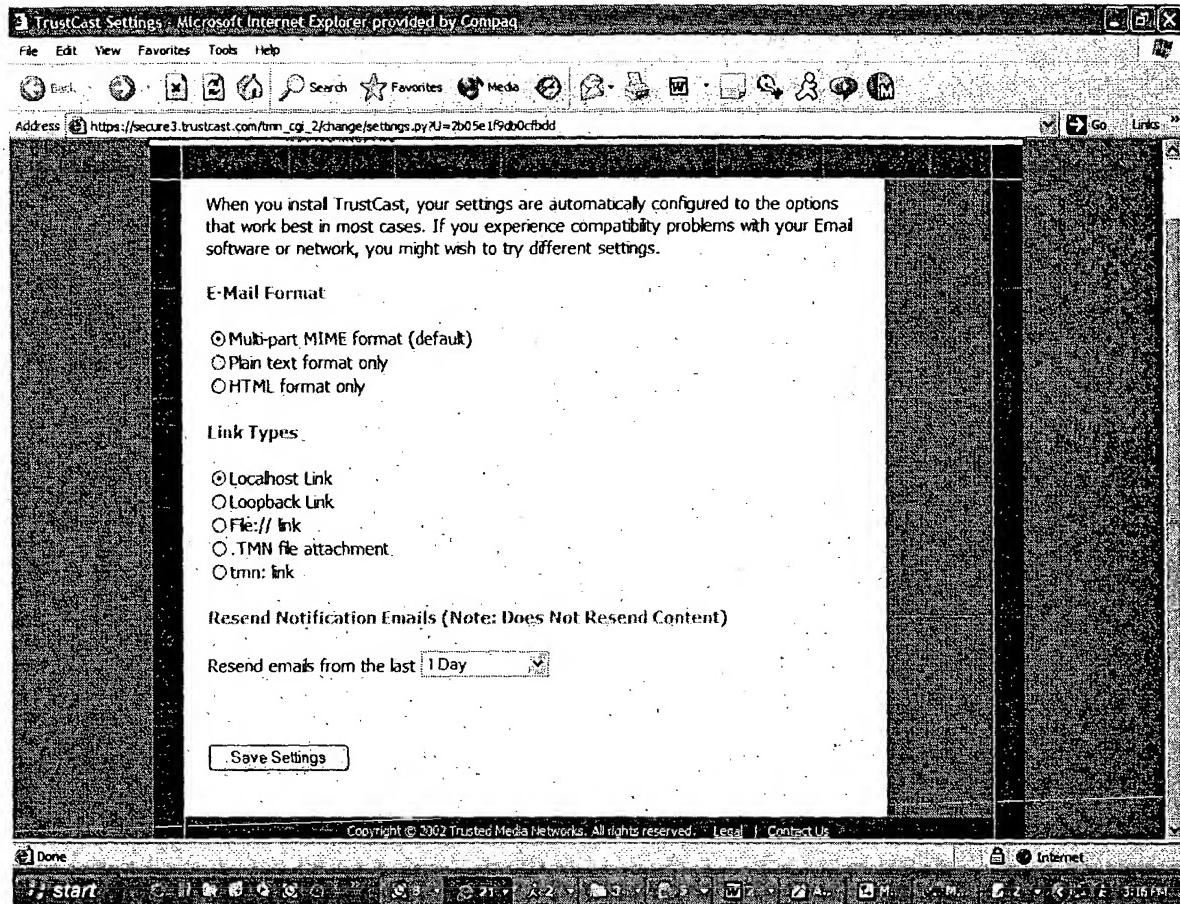


FIGURE 8A

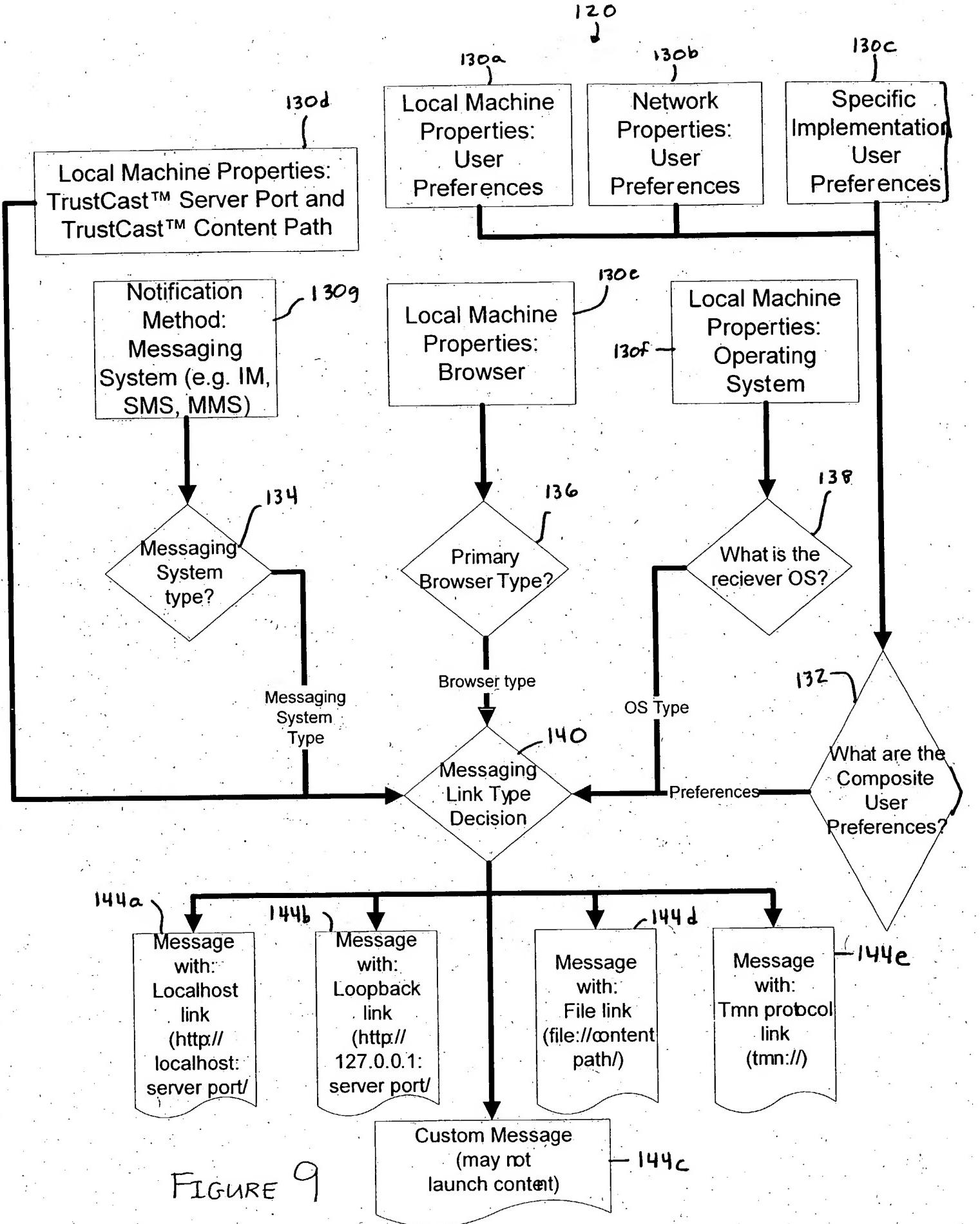
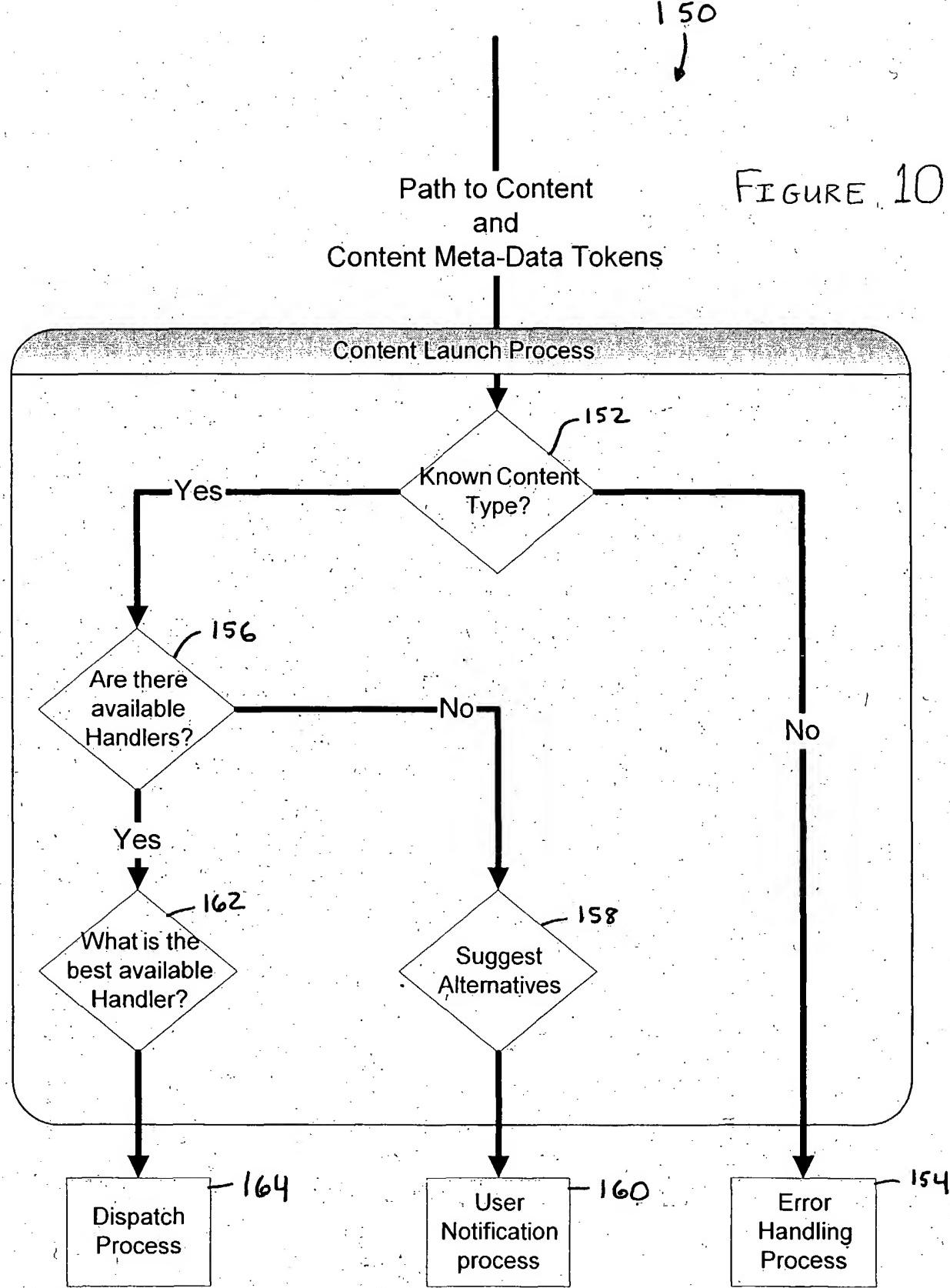


FIGURE 9

FIGURE 10



```
<key>goodPlayerList</key>
<array>
<dict>
    <key>extensions</key>
    <array>
        <string>.m3u</string>
    </array>
    <key>MIMEType</key>
    <string>audio/mpegurl</string>
    <key>helpers</key>
    <array>
        <dict>
            <key>name</key>
            <string>QuickTime Player</string>
            <key>fileCreator</key>
            <string>TVOD</string>
            <key>fileType</key>
            <string>M3U </string>
        </dict>
        <dict>
            <key>name</key>
            <string>RealPlayer</string>
            <key>fileCreator</key>
            <string>PNst</string>
            <key>fileType</key>
            <string>PNRA</string>
        </dict>
    </array>
    <key>description</key>
    <string>MP3 PlayList</string>
</dict>
<dict>
    <key>extensions</key>
    <array>
        <string>.mpeg</string>
        <string>.mp3</string>
    </array>
    <key>MIMEType</key>
    <string>video/mpeg</string>
    <key>helpers</key>
```

FIGURE 10A1

```
<array>
  <dict>
    <key>name</key>
    <string>QuickTime Player</string>
    <key>fileCreator</key>
    <string>TVOD</string>
    <key>fileType</key>
    <string>MPEG</string>
  </dict>
  <dict>
    <key>name</key>
    <string>RealPlayer</string>
    <key>fileCreator</key>
    <string>PNst</string>
    <key>fileType</key>
    <string>MPEG</string>
  </dict>
</array>
<key>description</key>
<string>MPEG-1 Movie</string>
</dict>
<dict>
  <key>extensions</key>
  <array>
    <string>.ASF</string>
    <string>.WM</string>
    <string>.WMA</string>
    <string>.WMP</string>
    <string>.WMV</string>
  </array>
  <key>helpers</key>
  <array>
    <dict>
      <key>name</key>
      <string>Windows Media Player 7</string>
      <key>fileCreator</key>
      <string>Ms01</string>
      <key>fileType</key>
      <string>ASF_</string>
    </dict>
  </array>
</dict>
```

FIGURE 10A2

```
</array>
<key>description</key>
<string>Windows Media File</string>
</dict>
<dict>
<key>extensions</key>
<array>
<string>.asx</string>
<string>.wax</string>
<string>.wmx</string>
<string>.wvx</string>
</array>
<key>helpers</key>
<array>
<dict>
<key>name</key>
<string>Windows Media Player 7</string>
<key>fileCreator</key>
<string>Ms01</string>
<key>fileType</key>
<string>ASX</string>
</dict>
</array>
<key>description</key>
<string>Windows Media File</string>
</dict>
<dict>
<key>extensions</key>
<array>
<string>.tmn</string>
</array>
<key>MIMEType</key>
<string>application/x-tmn-tmn</string>
<key>helpers</key>
<array>
<dict>
<key>name</key>
<string>TrustCast Helper</string>
<key>fileCreator</key>
<string>tmnH</string>
```

FIGURE 10A3

```
<key>fileType</key>
<string>xTMN</string>
</dict>
</array>
<key>description</key>
<string>TrustCast link file</string>
</dict>
</array>
```

FIGURE 10A4

```

// -----
// € getPathFromExeString
// -----
TMNStringPtr
getPathFromExeString(TMNStringPtr sExeString)
{
    if(sExeString == 0) return 0;
    TMNStringPtr sPath;
    // parse the executable path out of the string
    int firstChar = 0;
    if(sExeString->charAt(0) == '\\')
    {
        sPath = sExeString->substring(1, sExeString->indexOf(TMNStr
        ("\\.exe\\ "))+4);
    }
    else
    {
        sPath = sExeString->substring(0, sExeString->indexOf(TMNStr
        ("\\.exe "))+4);
    }

    return sPath;
}
// -----
// € checkVersion
// -----
bool
checkVersion(TMNDictionaryPtr handlerDict, TMNStringPtr
inExeString)
{
    try
    {
        // compare versions
        TMNStringPtr VersionReq = handlerDict->get(TMNStr
        ("versionLow")).as((TMNString*)0);
        if(VersionReq == 0 || pApp->verCmp(VersionReq, pApp-
        >getVersionFromFile(getPathFromExeString(inExeString))) <= 0)
        {
            VersionReq = handlerDict->get(TMNStr ("versionHigh")).as
            ((TMNString*)0);
    }
}

```

FIGURE 10B1

```

if(VersionReq == 0 || pApp->verCmp(VersionReq, pApp-
>getVersionFromFile(getPathFromExeString(inExeString))) >= 0)
{
    return true;
}
}
}
}
catch(...)
{
    // couldn't compare versions for some lame reason, assume true
    and hope
    return true;
}
return false;
}
// -----
// € isListed
// -----
bool
isListed(TMNStringPtr inHandler, TMNStringPtr inExeString,
TMNDictionaryPtr inList, bool userDefault)
{
    TMNDictionaryPtr handlerDict;

    // load the list of helper apps from the input dictionary
    TMNVectorPtr tempVec = inList->get(TMNStr
("helpers")).as((TMNVector*)0);
    if(tempVec == 0) return false;

    // check them all, if the handler matches assume true;
    TMNDictionaryPtr tempDict;
    TMNStringPtr tempString;
    for(int i = 0; i < tempVec->size(); i++)
    {
        tempDict = tempVec->elementAt(i).as((TMNDictionary*)0);
        if(tempDict == 0) continue;

        tempString = tempDict->get(userDefault ? TMNStr ("userKey") :
TMNStr ("rootKey")).as ((TMNString*)0);
        if(tempString == 0) continue;
}

```

FIGURE 10B2

```

if(inHandler->equalsIgnoreCase(tempString))
{
    handlerDict = tempDict->clone().as((TMNDictionary*)0);
    break;
}
}

// if something was found, check to make sure the version of the
// handler falls in the range in the list
if(handlerDict != 0)
{
    return checkVersion(handlerDict, inExeString);
}
return false;
}

// -----
// € getBestPlayerfromGoodList
// -----
TMNStringPtr
getBestPlayerfromGoodList(TMNDictionaryPtr inDict)
{
    // load the helpers from the list
    TMNVectorPtr helperVec = inDict->get(TMNString("helpers")).as((TMNVector*)0);
    if(helperVec == 0) return 0;
    TMNStringPtr goodPlayerKey;
    TMNDictionaryPtr tempDict;
    TMNStringPtr sFullExeString;
    TMNStringPtr sDefaultVerb;
    // for each helper listed
    for(int i = 0; i < helperVec->size(); i++)
    {
        // get the helper info
        tempDict = helperVec->elementAt(i).as ((TMNDictionary*)0);
        if(tempDict == 0) continue;
        sFullExeString = 0;
        // try to get the player based on the HKEY_CURRENT_USER info
        goodPlayerKey = tempDict->get(TMNString("userKey")).as
        ((TMNString*)0);
        if(goodPlayerKey != 0)
        {

```

FIGURE 10.B3

```

// get the path to the player
sDefaultVerb = pApp-
>getRegistryString(HKEY_LOCAL_MACHINE, TMNStr
("SOFTWARE\\Classes\\Applications\\") +goodPlayerKey + TMNStr
("\\shel"), NULL);
if(sDefaultVerb == 0)
{
    sDefaultVerb = TMNStr ("open");
}
sFullExeString = pApp-
>getRegistryString(HKEY_LOCAL_MACHINE, TMNStr
("SOFTWARE\\Classes\\Applications\\") +goodPlayerKey + TMNStr
("\\shell\\") + sDefaultVerb + TMNStr ("\command"), NULL);
}

// if if the HKEY_CURRENT_USER info does not exist, try the
HKEY_CLASSES_ROOT info
if(sFullExeString == 0)
{
    // get the path to the player
    goodPlayerKey = tempDict->get(TMNStr ("rootKey")).as
((TMNString*)0);
    if(goodPlayerKey == 0) continue;
    sDefaultVerb = pApp->getRegistryString(HKEY_CLASSES_ROOT,
goodPlayerKey + TMNStr ("\\shel"), NULL);
    if(sDefaultVerb == 0)
    {
        sDefaultVerb = TMNStr ("open");
    }
    sFullExeString = pApp-
>getRegistryString(HKEY_CLASSES_ROOT, goodPlayerKey +
TMNStr ("\shell\\") + sDefaultVerb + TMNStr ("\command"), NULL);
    // if nothing has been found, continue
    if(sFullExeString == 0)
    {
        continue;
    }
}

// check to make sure the file exists (hasn't been uninstalled)
TMNFilePtr fileExe = new

```

FIGURE 10B4

```

TMNFile(getPathFromExeString(sFullExeString));
if(!fileExe->exists()) continue;
// make sure that the versions of the file falls in the range specified
in the list
if(tempDict != 0 && sFullExeString != 0)
{
    if(checkVersion(tempDict, sFullExeString)) return sFullExeString;
}
}
return 0;
}
// -----
// € getApprovedPlayer
// -----
TMNStringPtr
TMNClient::getApprovedPlayer(TMNStringPtr inFileType)
{
    TMNProcLog procLog("TMNClient::getApprovedPlayer");
    TMNStringPtr sExtension = TMNStr ("..") + inFileType;
    bool userDefault = true;
    // get the default handler info from HKEY_CURRENT_USER
    TMNStringPtr sDefaultHandler = pApp-
>getRegistryString(HKEY_CURRENT_USER, TMNStr
("Software\\Microsoft\\Windows\\CurrentVersion\\Explorer\\FileExts\\\")
+ sExtension, TMNStr ("Application"));
    if(sDefaultHandler == 0 || sDefaultHandler->length() == 0)
    {
        // if there is none there, try HKEY_CLASSES_ROOT
        userDefault=false;
        sDefaultHandler = pApp-
>getRegistryString(HKEY_CLASSES_ROOT, sExtension, NULL);
    }
    // get the executable path
    TMNStringPtr sDefaultExeString;
    if(sDefaultHandler != 0)
    {
        TMNStringPtr sDefaultVerb;
        // if the info came from HKEY_CURRENT_USER, check the proper
        spot in the registry
        if(userDefault)

```

FIGURE 10B5

```

{
    sDefaultVerb = pApp-
    >getRegistryString(HKEY_LOCAL_MACHINE, TMNStr
    ("SOFTWARE\Classes\Applications\"") + sDefaultHandler + TMNStr
    ("\shel"), NULL);
    if(sDefaultVerb == 0)
    {
        sDefaultVerb = TMNStr ("open");
    }
    sDefaultExeString = pApp-
    >getRegistryString(HKEY_LOCAL_MACHINE, TMNStr
    ("SOFTWARE\Classes\Applications\"") + sDefaultHandler + TMNStr
    ("\shell\"") + sDefaultVerb + TMNStr ("\command"), NULL);
}

// if the info came from HKEY_CLASSES_ROOT, check the proper
spot in the registry
else
{
    sDefaultVerb = pApp->getRegistryString(HKEY_CLASSES_ROOT,
    sDefaultHandler + TMNStr ("\shel"), NULL);
    if(sDefaultVerb == 0)
    {
        sDefaultVerb = TMNStr ("open");
    }
    sDefaultExeString = pApp-
    >getRegistryString(HKEY_CLASSES_ROOT, sDefaultHandler +
    TMNStr ("\shell\"") + sDefaultVerb + TMNStr ("\command"), NULL);
}

// check to make sure the file exists (hasn't been uninstalled)
if(sDefaultExeString != 0)
{
    TMNFilePtr fileExe = new
    TMNFile(getPathFromExeString(sDefaultExeString));
    if(fileExe->isAbsolute() && !fileExe->exists())
        sDefaultExeString = 0;
}
}

// get the good/bad player lists
TMNVectorPtr goodVector = getGoodPlayerList();

```

FIGURE 10B6

```

TMNVectorPtr badVector = getBadPlayerList();
TMNDictionaryPtr goodPlayerList;
TMNDictionaryPtr badPlayerList;
// get the good player list for the current file type
if(goodVector != 0)
{
    TMNDictionaryPtr tempDict;
    TMNVectorPtr tempVec;
    TMNStringPtr tempString;
    bool found = false;
    for(int i = 0; i < goodVector->size(); i++)
    {
        tempDict = goodVector->elementAt(i).as ((TMNDictionary*)0);
        if(tempDict == 0) continue;

        tempVec = tempDict->get(TMNSTr
("extension")).as((TMNVector*)0);
        if(tempVec == 0) continue;
        for(int j = 0; j < tempVec->size(); j++)
        {
            tempString = tempVec->elementAt(j).as ((TMNString*)0);
            if(tempString == 0) continue;
            if(inFileType->equalsIgnoreCase(tempString))
            {
                found = true;
                goodPlayerList = tempDict->clone().as ((TMNDictionary*)0);
                break;
            }
        }
        if(found) break;
    }
}
// get the bad player list for the current file type
if(badVector != 0)
{
    TMNDictionaryPtr tempDict;
    TMNVectorPtr tempVec;
    TMNStringPtr tempString;
    bool found = false;
    for(int i = 0; i < badVector->size(); i++)

```

FIGURE 10B7

```

{
    tempDict = badVector->elementAt(i).as ((TMNDictionary*)0);
    if(tempDict == 0) continue
    tempVec = tempDict->get(TMNStr
    ("extension")).as((TMNVector*)0);
    if(tempVec == 0) continue;
    for(int j = 0; j < tempVec->size(); j++)
    {
        tempString = tempVec->elementAt(j).as ((TMNString*)0);
        if(tempString == 0) continue;
        if(inFileType->equalsIgnoreCase(tempString))
        {
            found = true;
            badPlayerList = tempDict->clone().as ((TMNDictionary*)0);
            break;
        }
    }
    if(found) break;
}
}

// if there is no mention of this filetype in the list, return the default.
if(goodPlayerList == 0)
{
    procLog.writeLog("No good player list specified for this filetype");
    // if there is no default, error.
    if(sDefaultExeString == 0)
    {
        procLog.writeLog("No default player available");
        logError("TMNClient::getApprovedPlayer", TMNStr ("Unable to
locate executable: "));
        return 0;
    }
    // make sure the default is not listed as bad
    if(badPlayerList != 0 && isListed(sDefaultHandler,
    sDefaultExeString, badPlayerList, userDefault))
    {
        procLog.writeLog("Default player listed as bad, no good list to
choose from");
        logError("TMNClient::getApprovedPlayer", TMNStr ("Default player

```

FIGURE 10B8

```
listed as bad"));
    return 0;
}

procLog.writeLog("Using default player");
return sDefaultExeString;
}
// no default player, check the good list to see if there are ANY
players installed
if(sDefaultExeString == 0)
{
    procLog.writeLog("No default player available");
    TMNStringPtr sExe = getBestPlayerfromGoodList(goodPlayerList);
    if(sExe != 0)
    {
        procLog.writeLog("Found good player");
        return sExe;
    }
}

// no good players installed.
procLog.writeLog("No good players installed");
logError("TMNClient::getApprovedPlayer", TMNStr ("No player
found"));
return 0;
}
// check good player list, if default is in there, run with it
if(isListed(sDefaultHandler, sDefaultExeString, goodPlayerList,
userDefault))
{
    procLog.writeLog("Default player in good list");
    // in case of a all-inclusive version in the good list, make sure the
version that
    // is the default is not on the bad list.
    if(badPlayerList == 0 || !isListed(sDefaultHandler, sDefaultExeString,
badPlayerList, userDefault))
    {
        procLog.writeLog("Using default player");
        return sDefaultExeString;
    }
}
```

FIGURE 10 B9

```
    procLog.writeLog("Default player also in bad list");
}

// if the bad player list is null, then ONLY play with a player from the
// good list
// if the handler is in the bad list, find the first installed player in the
// good list and return
if(badPlayerList == 0 || isListed(sDefaultHandler, sDefaultExeString,
badPlayerList, userDefault))
{
    procLog.writeLog("Exclusive good list detected");
    TMNStringPtr sExe = getBestPlayerfromGoodList(goodPlayerList);
    if(sExe != 0)
    {
        procLog.writeLog("Found good player");
        return sExe;
    }
    procLog.writeLog("No good players installed");
    logError("TMNClient::getApprovedPlayer", TMNStr ("No player
found"));
    return 0;
}
procLog.writeLog("Using default player");
// the handler is not bad nor is it good, return it and hope.
return sDefaultExeString;
}
```

FIGURE 10B10